E190AR: Lab 4 Report

Charles Dawson & Peter Johnson

***Abstract*— This report describes the work done to achieve level 1 automation (no automation) vision based control of the Romi Polulu using an "arUco" steering wheel. To achieve this end, we discuss fundamental concepts from computer vision, including the intrinsic and extrinsic calibration matrices and the Perspective-n-Place problem, before describing our implementation of vision-based control. Using a control strategy inspired by the video game Mario Kart, we were able to develop an intuitive, functional control system for our robot.**

I. Introduction

Images from optical sensors are some of the richest sources of data in robotics; in fact, humans are able to accomplish most of the tasks we would like robots to do simply using vision-based feedback. In this lab, we will explore basic image processing and computer vision techniques to implement a camera-based steering and control system for our robot. The end goal of this lab is to be able to drive the robot simply using a steering wheel marked with an arUco label. To this end, we will begin by discussing some of the theory behind computer vision, then discuss our implementation of the vision-based steering system, then conclude with a discussion of the experimental results of operating our robot using this system.

II. Theory

In this analysis, based on that in [1], we will rely on the pinhole camera model to describe our cameras. In this model, light from the object being imaged passes through a small pinhole and onto a sensor plane. The image that falls on the sensor plane is inverted by passing through the small hole, so it is common to describe pinhole cameras using their "image plane" rather than the sensor plane itself. The image plane (shown in Fig. 1) is the same distance f (the focal length) in front of the center of the camera as the sensor plane is behind the center, allowing us to perform calculations with the image plane rather than the (inverted) sensor plane.

If we consider an arbitrary point in space $PX(X, Y, Z)$, expressed in the camera frame as shown in Fig. 1, we can compute the projection of that point in the image plane $px(x, y)$, defined by the relation

$$Z\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where Z is factored out of the left-hand-side to allow us to express the transformation more concisely using linear algebra [1]. Note that the point px is expressed relative to an origin at the center of the image plane, but we would like it to be expressed relative to the bottom left corner of the image plane (as shown in Fig. 2), since that is how the pixels are referenced in the camera. In addition to shifting the offset, we also wish to express the coordinates in units of pixel rather than units of distance (like meter or centimeter), which we do using the conversion factors m_x and m_y, the pixel densities in the x and y directions (with typical units of pixels/centimeter). We do this by adding the translation by x_0 and y_0 to the focal length matrix above, then multiplying each x and y coordinate by the corresponding pixel density. We can combine these operations into a single transformation that relates an arbitrary point in space $PX(X, Y, Z)$ (expressed in the camera frame) to a pixel with coordinates (u, v) in the image plane:

$$Z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_x f & 0 & x_0 & 0 \\ 0 & m_y f & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

In particular, we refer to the non-trivial part of this transformation matrix as the "intrinsic matrix" of the camera:

$$K = \begin{bmatrix} m_x f & 0 & x_0 \\ 0 & m_y f & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

The intrinsic matrix is usually derived based on calibration data, as the pixel densities m_x and m_y, the focal length f, and the principal axis offsets x_0 and y_0 vary from camera to camera. Using this transformation, we can convert any arbitrary point (X, Y, Z) by carrying out the matrix multiplication in equation (1), then dividing

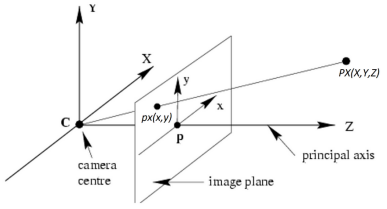

Fig. 1. Pinhole camera model, showing the image plane [1].

the result by Z to yield the location of the corresponding pixel in the image plane. Note that this transformation is not invertible, since the information on the depth of the original point is lost. In more complicated models of a camera, a skew term s is often introduced into the intrinsic matrix, yielding the modified intrinsic matrix

$$K_{skew} = \begin{bmatrix} m_x f & s & x_0 \\ 0 & m_y f & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note that the pixel densities and focal length are usually not calibrated independently, so only the product terms $m_x f$ and $m_y f$ are known.

In addition to the intrinsic matrix, which defines the transformation from a point in space in the camera's local frame to a point on the image plane, we also need the so-called extrinsic matrix to relate the camera's local coordinate frame to the global frame. Specifically, the extrinsic matrix is the homogeneous transformation from the global coordinate frame I to the camera frame C:

$$H_I^C = \begin{bmatrix} R_{11} & R_{12} & R_{13} & d_1 \\ R_{21} & R_{22} & R_{23} & d_1 \\ R_{31} & R_{32} & R_{33} & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

This extrinsic matrix can be estimated by solving a so-called "perspective-n-point" problem, wherein if the camera can see 3 points whose locations in the global frame are known, you can solve for the location and orientation of the camera (thus determining the extrinsic matrix) via triangulation. To solve this problem, it is vital that the intrinsic matrix of the camera is known beforehand (usually based on some calibration data). In our case, the calibration-data-based intrinsic matrix is contained in `/camera_info` ROS topic, so that data is needed to solve the PnP problem because it provides the intrinsic matrix.

The PnP problem is related to our application because we want to find the position and orientation of an arUco marker relative to our webcam. Because homogeneous transformations are invertible, this is equivalent to finding the homogeneous transformation from the arUco marker frame to the camera frame (i.e. the camera's extrinsic matrix in the arUco marker frame). This extrinsic matrix can be found by solving a PnP problem, where the 3 known points are 3 corners of the arUco marker, since the dimensions of the marker are known.

III. Implementation

The first step in the implementation of the arUco computer vision based control was calibrating our camera. ROS takes advantage of the webcam built into laptops through the `usb_cam` and `camera_calibration` packages. We wrote a ROS launch file to launch the camera node with the `auto_focus` flag set to true, overwriting the example which had just set the focus value to 50. The autofocus set to true is a sufficient setting since the focal length of the webcams in laptops will not change so any parameter changes are irrelevant. Once the camera node was running which was visible through the light next to the webcam and when the `/image` topic was added to rviz and visible, the next step was to calibrate the camera. This was as simple as running the calibration node which brought up a window which displayed our arUco calibration checkerboard overlaid with highlights as seen in Fig. 3.

The checkerboard was moved around the camera frame in a manner such that a full range of motion was covered. To achieve complete calibration in the x direction movement was needed left and right, y direction was up and down in the field of view, while to achieve calibration for size (z) the board had to move forwards and away from the camera, as well as tilting. Once enough data has been collected, the node is able to produce a calibration file. We can then look at the camera matrix and see that it is of the form we would expect from an intrinsic matrix. From here, a `/camera_info` yml file is produced which can then be used by the ROS camera drivers. Once the camera is calibrated, the cameracheck node is able to subscribe to the rectified images which will be used by arUco.

Now that the camera correctly produces images, arUco can be brought up to utilize them for control. The package we used was `aruco_detect`. For our launch file, we set the `dictionary` and `fiducial_length` argument

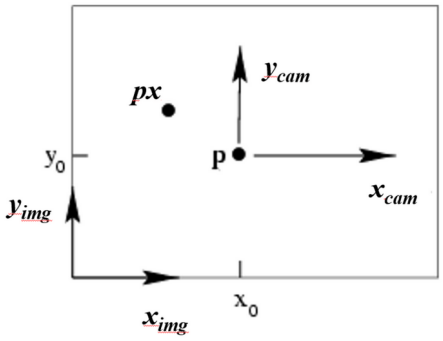

Fig. 2. Different coordinate systems on the image plane [1].

Fig. 3. Successfully registering arUco checkerboard

to be 1 and 0.104 respectively. Dictionary 1 corresponds to arUco markers in 4x4 grids with squares 0.1 m long. The `fiducial_length` argument describes the length of a fiducial in meters. This is the side length in one of the squares of our arUco checker grid. Once running, this publishes `/fiducial_msgs/FiducialTransforms`.

The final step in our vision based control of the robot is to write a control node subscribing to `/fiducial_transforms` and directly publishing `/cmd_vel` to the robot. A steering wheel based on arUco target was fashioned and used to control acceleration, turning and idling.

IV. Experiment

To drive our robot using an arUco marker, we constructed the arUco-based steering wheel shown in Fig 4. This wheel has padded grips, which help both with comfort and with preventing the user's hands from occluding any portion of the arUco marker. We define a control and steering system in which turning the steering wheel clockwise or counterclockwise makes the robot rotate in place in the same direction. We felt that this is an intuitive control scheme since it reflects how cars are driven in the real world. To move the robot forwards and backwards, the user tilts the steering wheel forward (to accelerate forwards) or backwards (to accelerate backwards), which we felt was intuitive because it matches the controls used in Mario Kart, a popular video game. Because of the familiarity of this control scheme, we feel that this system will be easy for new users to pick up with minimal training.

To actually compute the mapping between steering wheel motions and robot control inputs, we relied primarily on the Euler angles of the steering wheel's arUco marker relative to the webcam's coordinate frame. In particular, we set the angular rotation of the robot ω equal to the pitch angle θ. Since the arUco coordinate frame is defined with its y-axis facing towards the camera and its x-axis facing to the user's right, this means that the pitch angle θ corresponds to how much the steering wheel has been turned, either clockwise ($\theta < 0$) or counterclockwise ($\theta > 0$). Note that the sign of theta automatically makes the robot spin in the same direction as the steering wheel is turned.

To determine the linear speed of the robot, we rely on the roll angle ϕ. When the steering wheel is oriented pointing directly at the camera, $\phi = \pi$, which poses some difficulties, since the Euler angle ϕ is constrained to fall within the interval $(-\pi, \pi]$, so when the steering wheel is tilted slightly forward the roll angle ϕ will have a value close to $-\pi$, but when it is tilted slightly backwards then ϕ will be close ti $+\pi$. Thus, if the angle of steering wheel tilt is α (defined as positive when tilting forward and negative when tilting backwards), then the roll angle ϕ will be

$$\phi = \begin{cases} -\pi + \alpha & \alpha \geq 0 \\ \pi + \alpha & \alpha < 0 \end{cases} \tag{3}$$

To deal with this, we defined the following piecewise function to relate robot velocity v to roll angle ϕ:

$$v = \begin{cases} -\pi + \phi & \phi > 0 \\ \pi + \phi & \phi \leq 0 \end{cases} \tag{4}$$

By composing these two functions, we see that v is related to α by the simple relationship:

$$v = \begin{cases} -\pi + (\pi + \alpha) & \alpha < 0 \\ \pi + (-\pi + \alpha) & \alpha \geq 0 \end{cases} \tag{5}$$

$$v = \begin{cases} \alpha & \alpha < 0 \\ \alpha & \alpha \geq 0 \end{cases} \tag{6}$$

$$v = \alpha \tag{7}$$

Finally, to make control easier, we impose a deadband on the linear velocity v so that $v = 0$ is $\alpha < \pi/20$.

The steering wheel and `/mariokart` node proved to be effective at harnessing arUco to steer our robot. After first it was unwieldy to use, but with a bit of practice and going gingerly we were able to achieve control over the robot to the degree that we could navigate a figure-8 course around two water bottles without knocking them over. While our control scheme is sufficient, we do plan on improving it before the race. The main change will probably be changing acceleration from tilting the steering wheel towards the camera to its distance from the camera (moving close is faster, while backing up is slower). Turning will remain the same and stopping will remain the same. An additional change we are entertaining is the possibility of fixing the arUco marker to the drivers shirt, and instead moving the laptop relative to the marker. This would result in our controls being inverted, but that would be a simple fix.

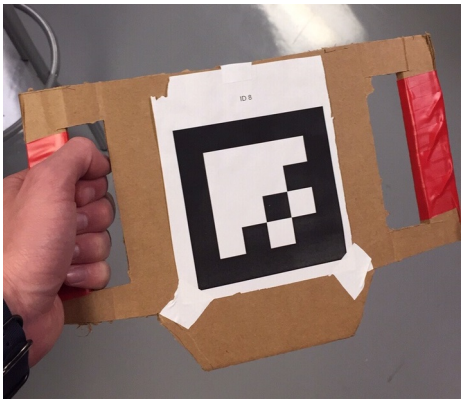

Fig. 4. arUco-based steering wheel

V. Conclusions

We were able to successfully implement level 1 autonomy of the Romi Polulu based on computer vision using arUco detect. arUco markers are detected correctly and we can perform fundamental robot control such as acceleration, deceleration, stopping and turning by manually operating a marker and extracting the fiducial transforms from its movement and converting them to /cmd_vel messages. This work helps us prepare for the final competition, which will involve one of us steering the robot around a course in Parsons using this vision-based control method.

References

[1] Y. Chang. E190AR. Class Lecture, Topic: "Point Tracking." Department of Engineering, Harvey Mudd College, Claremont, CA. Slides from G. Hua and P. Mordohai.